DOCUMENT RESUME

ED 276 439 IR 012 504

AUTHOR Larson, Jay
TITLE Problem Solving with Generic Algorithms and Computers.
PUB DATE Oct 86
NOTE 15p.; In: Thinking across the Disciplines. ISII 1986. Proceedings of the Annual Conference of the International Society for Individualized Instruction (15th, Atlanta, Georgia, October 9-11, 1986); see IR 012 493.
PUB TYPE Viewpoints (120) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Algorithms; *Computers; Data Processing; Educational Change; Flow Charts; *Problem Solving; Scientific Methodology

ABSTRACT

Success in using a computer in education as a problem-solving tool requires a change in the way of thinking or of approaching a problem. An algorithm, i.e., a finite step-by-step solution to a problem, can be designed around the data processing concepts of input, processing, and output to provide a basis for classifying problems. If educators concentrate on the how to, rather than on the correctness of the results, computers in the classroom can speed up the testing of specific algorithms. Developing an adequate, workable algorithm requires the critical steps of precisely defining the problem; careful attention to clarity, detail, and sequence; and a knowledge of the problem in order to explain the phases and smaller steps needed to solve the problem. The concepts of pseudocode and flowcharting standardize, solidify, and simplify the entire process. The ability to classify problems into one of three general categories (narratives, computational/verbal, and inputs/outputs) provides researchers with the framework upon which to build new and refine existing algorithms. Suggestions for the problem definition phase and an outline for this presentation are included. (DJR)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Jay Larson

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
[X] This document has been reproduced as received from the person or organization originating it.
[ ] Minor changes have been made to improve reproduction quality.
• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

ED276439

IR012504

PROBLEM SOLVING

WITH

GENERIC ALGORITHMS AND COMPUTERS

September 1986 BY Jay Larson, MBA

"People always get what they ask for;
the only trouble is that they never know,
until they get it, what it actually is that
they have asked for."

Aldous Huxley

Recently, there has been alot of effort spent on putting computers in the classroom. The main justification is that the computer is a problem-solving tool. It is well understood that a computer is best utilized to solve repetitive problems requiring the manipulation of large amounts of data and fast processing with a high degree of accuracy. A series of instructions developed to enable the computer to solve a particular problem is called a program. Programmers are persons responsible for giving the computer these instructions in a specific language. They must derive the solution to a given problem and then insure that these instructions are coded so that the computer can carry them out.

The actual writing of the program, however, is only a part of the larger problem-solving methodology. Programs

represent solutions to problems, which are developed by using algorithms. An algorithm is a finite, step-by-step solution to a problem. Algorithms may be narrative outlines of steps needed to solve the problem, like recipes, or they may be composed of pictoral diagrams. Usually programmers will use either psuedocode or flowcharts to detail these solutions. The method used is not nearly as important as the ability to define precisely what the actual problem is and detail exactly in definite steps what needs to be done to solve the problem.

A concise, overall problem-solving strategy was first suggested by George Polya in 1945. (1) All authors on the subject of problem-solving agree that the most critical step is the first one - definition of the problem. It is this step that requires the most imagination, ingenuity, and creativity. The problem must be carefully examined to identify clearly what information must be produced to solve the problem. Then the problem solver must identify the information given in the problem which may be useful in obtaining the solution. Finally, since the computer is only a machine that has no inherent problem-solving capabilities, a procedure to generate the desired results from the given inputs must be formulated as a detailed sequence of simple steps. An outline for this problem definition phase is suggested as follows:

1. Define the problem
    a. Clearly describe the problem

b. Describe what the output of a succesful solution should be and how it would look

2. Develop the solution
   a. Specify precisely the inputs (the data to be used) and the outputs (the results) in a finite number of steps
   b. Divide the problem into small definite, crystal clear steps which tell exactly what to do
3. Organize, code and document the program

Success in using a computer as a problem solving tool requires a change in the way of thinking or of approaching a problem. Yet the steps used to develop a program roughly parallel the steps in real-life situations. For example, suppose a cook wants to make a dessert. An experienced cook would not often start with a canned recipe; instead he/she would start with the general problem of what to cook. After answering the question of what dessert to make, the cook would decide on the ingredients and follow the steps of the selected recipe. Despite the similarity, a cook's finished recipe is usually not an algorithm though because it is often vague about the input and output; i. e. a cup of nuts as compared to 5,022 peanuts or a batch of cookies as contrasted to 223 cookies. An algorithm must contain precise descriptions of the data and their relationships. It is vitally important that the computer user know the output requirements before the algorithm can be developed. Entering

test data into the algorithm allows us then to test its validity.

Normally, algorithms provide a computer programmer with the logic needed to arrive at the solution to a computer problem. Their development force the programmer to understand the problem better and to set forth the output objectives.(2) They can be used also to indicate areas where more answers are needed. For instance, a business manager wants a program to keep track of items in inventory. The problem from the user´s point of view is keeping track of inventory, providing timely and accurrate reports, and easy to use. However, what the programmer must now do is take these specifications and determine whether or not a computer can solve the problem, and if it can, redefine the problem in such a way that it (1) satisfies the user´s requirements and (2) can be solved by a computer program. Nothing can be assumed. The outputs must be precisely specified; the data needed to produce the desired output must be specified; the formulas or processes needed to solve the problem must be specified. A problem that is not defined precisely, correctly and completely is a problem for which there is no computer solution.

In general, the relationship between algorithms and computers is this: you can almost never solve a problem with a computer without an algorithm, but you can use a algorithm to solve a problem even if you have no computer available. The computer takes everything literally and knows only what

it has been told. An algorithm has in fact been defined as a logical sequence of unambiguous operations that when carried out lead to the solution of the specified problem. English is full of ambiguities; alot of words have more than one meaning or shades of meaning that are susceptible to different interpretations. A "heaping teaspoon" may be ok for a recipe but not for a computer program. Also, the computer is very logical and will do things in the exact order in which it is told to do them. Additionally, most problems require careful attention to sequence if they are to be solved, and the algorithm must show the correct sequence. In fact all of the steps even the obvious ones must be included in the algorithm. The computer does not assume anything, even the obvious. For example, a recipe might say "mix the ingredients". A robot (a computer) must be told to "turn on the mixer" prior to being told to mix the ingredients.

Developing an adequate, workable algorithm requires a great deal of attention to detail and a knowledge of the problem. We must ask detailed questions about all the elements of the problem and have taken nothing for granted. Thus, the persons who know the most about the problem are in the best position to develop its solution. If a person can explain the steps used to solve a problem, then that person can use a computer to solve the problem; however, if a person cannot explain the solution to another, then a computer could not be used. Computers don´t understand "intuition" or "gut

feelings".

Polya (1945) described what he called the inventor´s paradox. "Sometimes a larger problem is easier to solve than a simpler one. The more ambitious plan may have more chance of success."(3) In other words, it is easier to solve a more difficult problem rather than a simpler one because seeing one problem in context of a lager one often leads to the solution of a simpler one. Once we have an overview of the problem, then the "divide and conquer" problem-solving strategy can come into play. Nearly every problem is composed of several phases made up of smaller steps each consisting of inputs, processing functions and outputs.

For instance, let´s suppose that we´ve been asked to solve the following problem. A teacher would like to use the computer in place of his/her gradebook; that is, he/she would like a program that would post and average all the grades of a particular test, produce as output the class average, the number of students attaining each grade and the total number of failures. To solve this problem, one must begin with the output, deciding what data is necessary to generate the average and the determining the scores needed for each grade. A register or counter must be setup to keep track of the number of students whose scores fall within these ranges, esp. the number of failing scores. One must also know the formula for calculating an average (A=SUM/Total # of items). Also, the problem-solver must determine the number of questions which will be on the tests, in order to calculate

the percentage scores to determine the grades. Also he/she must determine whether the instructor grades on a fixed percentage basis or on a curve, which would change the ranges of scores for each grade. In other words, a relatively simple problem can become complicated as the requirements for each of the smaller pieces are developed in terms of input, processing and output. The need for clarity and proper sequencing is apparent as one tries to develop the algorithm or explain to someone who knows nothing about grading or averaging how to create a gradebook. The computer solution cannot assume anything. Actually, the person who knows the most about calculating grades, which may be different than the person who is the most accurate at calculating grades, is in the best position to develop the algorithm because he/she knows the procedures, formulas and correct sequence. The development of the valid algorithm results from knowing what inputs and processing functions are required; consequently, the correct output will always result.

Here is our first indication of the direction that education should take - concentration on the how to, rather than on the correctness of the result. Machines in the classroom can speed up the testing of specific algorithms. It is a very subtle difference but one of great significance. Concentrating, believing that if the problem solving steps are properly determined and arranged that a correct answer will be generated every time the algorithm is used no matter where the data is from. Conversely, a result generated by

an inaccurate algorithm could never be trusted. The educational controversy and implication become apparent as we concentrate on the algorithms instead of the outcomes. Should we have each generation of students recreate the discovery of particular algorithms, or should a package of algorithms be presented to each student as their grade-level and understanding advance? Would students "appreciate" algorithms generated by earlier generations? It would seem that the computer offers educators not only the ability to rapidly detect faulty problem-solving strategies but also the opportunity to explore and refine the valid algorithms of the past. Aren't these the aims of traditional (those without computers) education programs - respect for the past and improvement for the future? Where does problem-solving fit in the scheme of educational goals? Are educators committed to the proposition that a student who has been trained in problem solving can cope the better after graduation than those who have not been so trained? In short, do problem solvers make the best citizens? These are questions that each educator must answer individually before proceeding.

Several problem solving techniques have been used by educators throughout the years. Perhaps one of the oldest is called the scientific method. Development of an hypothesis and the establishment of various alternatives for testing is central to this method. The approach finds validity in a laboratory setting but is almost impossible to computerize

because of its "discovery" notion; that is, the outcome is unknown at the time the testing begins. The hypodeductive reasoning method of problem solving also has little application in a computer-type environment because again of its outcome uncertainty. Many problems aren't what they seem to be at first glance. Many problem solvers of the past have used none of these methods but claim to be "guided by intuition" - the answers just came. This is another method that finds little applicability in our computerized society. Only algorithms, their development and their use, not only facilitate the problem-solving process but also allow us to create computer solutions which can be stored, shared and scrutinized. The concepts of psuedocode and flowcharting standardize, solidify and simplify the entire process. Persons who have solved previous problems now have a mechanism that permits them to convey to future generations procedures, processes and practices which may have taken years, even generations, to refine and perfect. The algorithm technique meets the needs of this generation; none of the others could.

Are the problems facing our students significantly different from those encountered by previous generations? If one looks at the small parts, the answer is definitely yes. However, a look at the "big" picture reveals that many problems are just restatements of a previous type. For instance, suppose we were required to put together a jigsaw puzzle. The strategy might be to put all the pieces on a

table, look for all the border pieces, separate all pieces of the same color, put the border together and then compare the common colored pieces to the border pieces and fit them together. This same algorithm could be used with all jigsaw puzzles regardless of the number, sizes, shapes, or texture of the pieces. Laying out the algorithm actually helps to classify the problem.

As I have examined the "world" of problems, it seems there are but three major types with contrasting styles within each. The first category, I call the "storybook problems". Here we find the narrative-type problems - tell me how you did that. Our example of the jigsaw puzzle is in this category. The algorithms which solve these problems are either linear (proceed sequencially through one or more series of input, processing, output phases), or use the story tree format (branch through a series of input, processing, output phases based on certain conditions). The solutions to these problems require creativity and imagination. They are probably the most challenging for computer advocates and programmers. The next category, I call the "mathbook problems". As any high school student knows, the math problems come in two types - computational and verbal. The story problems are the least preferred, I think simply, because the algorithm is not given and must be devised. Traditionally, students spend more time solving computational type problems than they spend learning how to develop algorithms from story problems. I've noticed that usually

the computational problems in a math textbooks out number the story problems by as much as ten to one. Ironically, many educators prohibit their students from using computers/calculators to solve computational problems because they can complete them very fast and without errors. The story problems are usually assigned as extra credit, which means few students try to set up the algorithms (equations) to solve them, or even read the problems. The third type I call the "handbook problems". These look primarily at the inputs/outputs required by the problem. All programmers know that computer languages speak in terms of variables and constants and that numeric and character string variables or constants must be handled differently. Problem solvers must look at the data and its source to determine whether it will be variable or constant. As one develops the algorithm to solve the problem, this condition becomes apparent and the algorithm can be modified to meet these conditions. Nonetheless, the value of being able to classify life´s problems into one of these three general categories provides a researcher with the framework upon which to build new and refine existing algorithms.

FOOTNOTES

1. George Polya, HOW TO SOLVE IT (New Jersey: Princeton University Press, 1945), pp. 5-6.

2. James F. Peters III, PROBLEM SOLVING WITH PASCAL (New York: CBS College Publishing, 1986), p. 13.

3. Polya, op. cit., p. 121.

RECOMMENDED COROLLARY MATERIAL

Brightman, Harvey J. PROBLEM SOLVING: A Logical and Creative Approach. Georgia: Georgia State University, 1980.

Polya, George. MATHEMATICAL DISCOVERY: On Understanding, Learning, and Teaching Problem Solving. New York: John Wiley and Sons, Inc., 1962.

PROBLEM SOLVING WITH GENERIC ALGORITHMS AND COMPUTERS

SUMMARY:

Almost every problem is best solved by writing out a step-by-step solution. This is an algorithm. Many problems can be solved by those same steps, if they are written following the data processing steps -input, processing and output. Educators who are dedicated to the purpose of providing students with life-long foundations for problem solving would be able to teach these skills if they only knew how to classify daily problems into one of a limited number of algorithms. Students would then be able to expand their own creative thinking skills by generating variations of these to meet their everyday requirements of managing themselves, their time and other persons. This paper presents the necessary framework/ideas to cause this conceptual problem solving technique become a reality.

METHOD OF PRESENTATION:

Overhead slides with discussion and sample problems.

TARGET AUDIENCE:

Educators of students - 7th grade to adult.

EQUIPMENT NEEDED FOR PRESENTATION:

Overhead projector and blackboard.

OUTLINE

I. INTRODUCTION
   1. Types of algorithmns
      a. narrative
      b. pictoral
   2. Problem definition
      a. First step to success
      b. Not very easy
      c. Stated or assumed

II. ALGORITHMN DEVELOPMENT
   1. What it means to "really" understand the problem
   2. Regular use of algorithmns
   3. What can be assumed?
   4. What does it mean to "think logically"?
   5. Could you tell a robot where to go and what to do?
   6. Why tell someone else what you know how to do well?

III. INPUT, PROCESSING AND OUTPUT STRUCTURE
   1. Output the beginning of the end.
   2. What data would provide the desired results?
   3. What can the computer do to help?
   4. Identification/placement of the symbols
   5. Testing the algorithmn

IV. PROBLEM SOLVING TECHNIQUES
1. The "scientific" method
2. Hypodeductive reasoning and problem solving
3. Algorithmns
4. Intuition - what is it? Can it be passed to another?
5. Putting it in words or understandable symbols

V. CLASSIFYING PROBLEMS
1. Importance of the "big" picture
2. Linear vs story tree solutions
3. Verbal vs computational
4. Variables vs constants

VI. PUTTING IT ALL TOGETHER - SAMPLES TO TRY
1. Direction problems
2. Mathematical problems
3. Classification/sorting problems
4. The classical algorithmn
5. The computerized solution
6. People helping people